

# Python Summary 6: Format strings

## String basics:

Strings are made up of a list of characters. In Python 3, strings are made up of Unicode characters. For a detailed description of the reasons for this, and other topics covered in this document, see <http://www.diveintopython3.net/strings.html>

We have already seen that strings can be joined together (concatenated) by using the '+' operator. For example, if we wanted to print out a message saying who won a game, and what the score was we could do the following:

```
string_to_print = "The final score was " + str(player_1_score) + " - " + str(player_2_score) + ".  
Congratulations " + winner_name + "!"  
print(string_to_print)
```

To get a string similar to:

"The final score was 12 - 5. Congratulations Justin!"

## Format strings:

Combining strings using '+' works well when you only have a couple of pieces to join together. However, when things get complicated like in the string above it becomes difficult to see what the final string will look like.

To solve this issue, format strings were developed. Here is the example above rewritten using a format string:

```
message_template = "The final score was {0} - {1}. Congratulations {2}!"  
string_to_print = message_template.format(player_1_score, player_2_score, winner_name)  
print(string_to_print)
```

It is much easier to see from the message\_template what the final string will look like. Each {X} is a placeholder. When the format() function is called on the string, these placeholders are replaced with the values given to the function. {0} will be replaced with the first value (player\_1\_score in the example above), {1} with the second value, {2} with the third value and so on.

There are two ways to do this. If you want to use the same format multiple times (with different data) you should store the format string as a variable and call .format() on that variable each time you want to use it. This is what was done in the example above. Alternatively, you can call .format() directly on the string as you create it:

```
string_to_print = "The final score was {0} - {1}. Congratulations {2}!".format(player_1_score,  
player_2_score, winner_name)  
print(string_to_print)
```

Using a format string, you do not need to convert int or bool values into str values before they can be added to the string. The format function takes care of the conversion for you!

## Test Yourself:

Use format strings to print the following:

1. Ask the user for their name and age, then print a message including both pieces of information. (E.g. "Yay! Bob is 57!")
2. Ask the user for two numbers, and print out a message in the following format: "The sum of {0} and {1} is {2}" (e.g. "The sum of 2 and 7 is 9")

## More Complicated Formatting

Format strings can be powerful tools for laying out text. There is a Format Specification language, described in the python docs at <https://docs.python.org/2/library/string.html#format-specification-mini-language>, can be used to specify the number of digits or decimal places a float or integer should be printed with or the amount of space a string should be padded with. Here are some examples, with an explanation of how they work:

```
"{0:.2f}".format(3.1415926)
```

This will return "3.14". The 0 in the braces indicates that the first (and, in this case, only) argument will be put into the string. ':' is used to separate the formatting details from the index. The format specification of '.2f' tells python to use 2 decimal places for the float. This cannot be used with integer arguments.

```
"{0:>10}".format("Test")
```

This will return the string " Test" (6 spaces before the T). '>' tells python to right-align the text in the field, and the '10' specifies that the string should be 10 characters long. By default, Python uses a space to fill in any extra characters, but you can specify a different character to use.

```
"{0:^10}".format("Test")
```

This will return the string "\*\*\*\*Test\*\*\*\*". '\*' in this format is the character to fill extra space with. '^' tells python to center the string in the available space.

## Even More Detail

All of this is described in detail in the Python Documentation. Everything you can do with format specifiers (the bit after the ':') can be found by reading the Format Specification Mini-language documentation at <https://docs.python.org/3/library/string.html#formatspec>